dot commands

- `.help` → Show all available dot commands
- `.tables` → List all tables in the current database
- `.schema` → Show SQL used to create **all tables**
- `.schema table_name` → Show schema of a **specific table**
- `.headers on` → Show column names in query output
- `.mode column` → Pretty, aligned table output (best for learning)
- `.mode list` → Simple pipe-separated output
- `.nullvalue NULL` → Display `NULL` values clearly
- `.open filename.db` → Open or create a database
- `.read file.sql` → Execute SQL from a file
- `.quit` or `.exit` → Exit SQLite

1 Create table

```
CREATE TABLE students (
    roll_no INTEGER PRIMARY KEY AUTOINCREMENT,
    name    TEXT NOT NULL,
    class   INTEGER NOT NULL,
    dob     DATE NOT NULL
);
```

2 Insert sample data (20 students)

```
INSERT INTO students (name, class, dob) VALUES
('Aarav',   8, '2005-01-01'),
('Ishaan',  8, '2005-01-02'),
('Riya',    8, '2005-08-03'),
('Ananya',  8, '2004-01-01'),
('Kabir',   8, '2004-08-02'),

('Rahul',   9, '2004-01-03'),
('Sneha',   9, '2004-08-01'),
('Arjun',   9, '2003-01-02'),
('Priya',   9, '2003-08-03'),
('Kunal',   9, '2004-08-02'),

('Neha',   10, '2003-01-01'),
('Aditya', 10, '2003-01-02'),
```

```
('Pooja',  10, '2003-08-01'),
('Rohan',  10, '2004-01-03'),
('Simran', 10, '2004-08-02'),

('Vikas',   8, '2005-01-03'),
('Meera',   9, '2004-08-03'),
('Sahil',  10, '2003-08-02'),
('Tanya',   9, '2004-01-01'),
('Nikhil',  8, '2005-08-01');
```

3 Verify

```
SELECT * FROM students;
```

CREATE TABLE — general syntax

```
CREATE TABLE table_name (
    column1 datatype [constraints],
    column2 datatype [constraints],
    ...
);
```

Common constraints

- `PRIMARY KEY`
- `AUTOINCREMENT` (SQLite: only with `INTEGER PRIMARY KEY`)
- `NOT NULL`
- `UNIQUE`
- `DEFAULT value`

INSERT — single row

```
INSERT INTO students (name, class, dob)
VALUES ('Amit', 9, '2004-01-01');
```

INSERT — multiple rows

```
INSERT INTO students (name, class, dob)
VALUES
    ('Ravi', 8, '2005-08-01'),
    ('Sita', 9, '2004-01-02'),
    ('Mohan', 10, '2003-08-03');
```

Notes (important)

- Column order must match value order
- You can omit columns with:
  - `AUTOINCREMENT`
  - `DEFAULT`

SQLite storage model

- Dates in SQLite are stored as `TEXT` (`YYYY-MM-DD` recommended)
- `DATE` is **just a type name**
- Value is still stored as **TEXT** (`'YYYY-MM-DD'`)

SQLite has only **5 storage classes**:

- `NULL`
- `INTEGER`
- `REAL`
- `TEXT`
- `BLOB`

Everything else (`DATE`, `VARCHAR(100)`, `BOOLEAN`) is **mapped** to these.

```
SELECT dob, typeof(dob) FROM students;
```

Rule of thumb

- Writing `DATE` → improves **readability**
- SQLite still stores → `TEXT`

Type affinity

Type affinity = the **preferred storage class** SQLite tries to use for a column.

The 5 SQLite affinities

Affinity	What SQLite tries to store
`INTEGER`	Whole numbers
`REAL`	Floating-point numbers
`TEXT`	Strings
`NUMERIC`	Numbers if possible, else text
`BLOB`	Raw bytes

How SQLite decides affinity

SQLite looks at the **declared column type name**.

Examples:

- `INT`, `INTEGER` → `INTEGER`
- `CHAR`, `VARCHAR`, `TEXT` → `TEXT`
- `REAL`, `FLOAT`, `DOUBLE` → `REAL`
- `DATE`, `BOOLEAN`, `NUMERIC` → `NUMERIC`
- No type → `BLOB`

How constraints are written (syntax)

```
column_name datatype CONSTRAINT1 CONSTRAINT2 ...
```

or at **table level**:

```
CONSTRAINT constraint_name constraint_definition
```

Column-level constraints (most common)

```
id INTEGER PRIMARY KEY AUTOINCREMENT
```

Order doesn't matter (logically), but this is conventional.

Common column constraints

- `PRIMARY KEY`
- `AUTOINCREMENT` (SQLite-specific)
- `NOT NULL`
- `UNIQUE`
- `DEFAULT value`
- `CHECK (condition)`
- `REFERENCES table(column)` (foreign key)

Examples (column-level)

```
name TEXT NOT NULL
```

```
email TEXT UNIQUE NOT NULL
```

```
age INTEGER CHECK (age >= 0)
```

```
created_at TEXT DEFAULT CURRENT_DATE
```

Table-level constraints (used for combinations)

```
CREATE TABLE enrollments (
    student_id INTEGER,
    course_id  INTEGER,
    PRIMARY KEY (student_id, course_id)
);
```

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    email TEXT,
    CONSTRAINT unique_email UNIQUE (email)
);
```

FOREIGN KEY example

```
student_id INTEGER REFERENCES students(roll_no)
```

⚠ Must enable explicitly:

```
PRAGMA foreign_keys = ON;
```

PRAGMA

PRAGMA = SQLite **special command** to **configure or query database behavior**.

Think of it as:

> *Database settings & internal info*

Table info (columns, types, constraints)

```
PRAGMA table_info(students);
```

→ column name, type, NOT NULL, default, PK

Enable foreign keys (VERY important)

```
PRAGMA foreign_keys = ON;
```

→ enforce referential integrity

1 SELECT — general syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column1, column2, ...
HAVING group_condition
ORDER BY column1 ASC|DESC
LIMIT count OFFSET offset;
```

Execution order (important):

```
FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY → LIMIT
```

2 WHERE conditions

Comparison operators

```
=  !=  <  <=  >  >=
```

Logical operators

```
AND
OR
NOT
```

IN

```
WHERE class IN (8, 9)
```

BETWEEN (inclusive)

```
WHERE dob BETWEEN '2004-01-01' AND '2005-12-31'
```

LIKE (pattern matching)

```
WHERE name LIKE 'A%'   -- starts with A
WHERE name LIKE '%a'   -- ends with a
```

3 LIMIT

```
LIMIT 5
LIMIT 5 OFFSET 10
```

SQLite shorthand:

```
LIMIT 10, 5   -- OFFSET 10, LIMIT 5
```

4 ORDER BY (sorting)

Single column

```
ORDER BY dob ASC
ORDER BY name DESC
```

Multiple columns

```
ORDER BY class ASC, dob DESC
```

- Sort by `class` first
- Then `dob` within each class

5 GROUP BY

Used with **aggregate functions**:

```
COUNT()
SUM()
AVG()
```

```
MIN()
MAX()
```

Example

```
SELECT class, COUNT(*) AS total_students
FROM students
GROUP BY class;
```

Rule:

- Every selected column must be:
    - in `GROUP BY`, or
    - inside an aggregate function

6 Filter groups — HAVING

`WHERE` → filters rows `HAVING` → filters groups

```
SELECT class, COUNT(*) AS total
FROM students
GROUP BY class
HAVING COUNT(*) > 5;
```

7 Combined example

```
SELECT class, COUNT(*) AS total
FROM students
WHERE dob < '2005-01-01'
GROUP BY class
HAVING total >= 3
ORDER BY total DESC
LIMIT 2;
```